Exploring Generative Capabilities of Diffusion-based Deep Learning Models

COMP3547 Deep Learning Assignment 2022/2023

Piotr Borowiecki (svmm25)
piotr.borowiecki@durham.ac.uk

Abstract

This paper presents two implementations of diffusion-based deep generative models, designed to synthesize unique images, which could plausibly come from a training data set. The underlying theory is first discussed, followed by methodology, and empirical evaluation of results. Presented models are largely re-implementations of two existing papers [1, 2]. Training was performed using two data sets, CIFAR-10 [3] and FFHQ [4], on images of size 32x32 and 96x96 respectively. Results confirm that such models are capable of generating realistic, diverse, high-quality images, although relatively long training times are required to achieve this.

1 Introduction

Generative modelling can be seen as one of the most fundamental problems in deep learning, aimed at designing and developing algorithms for generating novel data samples representative of a given data set. The overall goal is to learn the underlying probability distribution, which could then be used to draw new images. The so called Generative Learning Trilemma, a trade-off between the sampling speed, mode coverage (diversity), and sample quality, has been a running problem and remains largely unresolved. The trilemma arises, since improving one of these goals typically comes at the cost of decreasing one or both of the others. Recently, diffusion-based models have emerged as a promising approach to achieve all three goals. These architectures use partial differential equations to describe the diffusion of a noise signal through a deep neural network, and can produce diverse and high-quality samples, while also accurately modeling the data distribution. This project serves as a starting point for further exploration. Implementations described in this report and submitted as part of the assignment are inspired by several sources, such as [5, 6, 7, 8, 9, 10, 11]. It is particularly inspired by and based on the work of Yang Song, published on his blog, and in accompanying Google Colab notebooks [9].

2 Background

2.1 Denoising Diffusion Probabilistic Model (DDPM)

The central concept in DDPMs involves gradually breaking down the patterns within a given data distribution using a forward diffusion process. This is followed by training a reverse diffusion process that re-establishes the original structure, resulting in a versatile generative model. By appropriately training the noise removal network, $\epsilon_\theta(x_t, t)$, random noise is eliminated until a unique new image appears, resembling those in the training data. The forward diffusion process is defined as $q(x_t \mid x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}\right)$, where $\mathcal{N}$ is the Normal Distribution, x_t is the output image at time t, $\sqrt{1-\beta_t}x_{t-1}$ is the mean, and $\beta_t I$ is the variance. The reverse diffusion process is defined as $p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$, with $p_\theta(x_{0:T}) = p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$, and $p_\theta(x_0) = \int p_\theta(x_{0:T})\, dx_{1:T}$, where θ are the parameters we train. The variance can be fixed to a certain schedule, meaning that only $\mu_\theta(x_t, t)$ must be learned. In terms of the objective function,

$-\log(p_\theta(x_0))$ cannot be used implicitly. This is because $p_\theta(x_0)$ is not easily tractable in practice, since it depends on all previous time steps. A variational lower bound may be computed instead as $-\log(p_\theta(x_0)) \leq -\log(p_\theta(x_0)) + D_{KL}(q(x_{1:T} \mid x_0) \| p_\theta(x_{1:T} \mid x_0))$, taking advantage of the Kullback-Leibler Divergence, which is a measure of how similar two distributions are. It is always non-negative, and defined as $D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$. The overall setup resembles Variational Autoencoders. By using algebraic manipulations, log rules and the Reparametrization Trick, the objective can be re-written as

$-\log(p_\theta(x_0)) \leq \sum_{t=2}^{T} D_{KL}\left(\mathcal{N}\left(x_{t-1}; \widetilde{\mu}_t(x_t, x_0), \widetilde{\beta}_t I\right) \| \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))\right) - \log(p_\theta(x_0 \mid x_1)$, where $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0$ and $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{a}_t} \cdot \beta_t$. Unfortunately, the page count limit does not allow for presenting the full derivation, but the loss in this context can be written as $L_t = \frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\hat{\alpha}_t)} \|\epsilon - \epsilon_\theta(x_t, t)\|^2$. Ho et al. [1] found that discarding the coefficient increases sample quality, and hence, the loss function is eventually given as $L_t = \|\epsilon - \epsilon_\theta(x_t, t)\|^2$. Sampling from $q(x_t \mid x_0)$ is governed by $q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha}_t}x_0, (1-\overline{\alpha_t})\mathbf{I})$, wheres sampling from $p_\theta(x_{t-1} \mid x_t)$ can be formalised as $p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 \mathbf{I})$. Here, $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$, and $\epsilon_\theta(x_t, t)$ is our neural network.

2.2 Score-matching Model (SMM)

The main idea behind score-matching appears to be originating from [12], and is such that instead of estimating the probability density function itself, its gradient may be utilized, where the gradient of a probability density function with respect to the input dimensions, $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, is called the score. The gradient and the underlying density function are related to each other and thus, MMMs are trained to estimate it. As opposed to many traditional approaches, computing the score does not require Z_θ and is therefore tractable. Before samples can be generated, a diffusion process takes place, where initial data is gradually corrupted and perturbed with noise to the point, where it becomes just noise. The process is based on the stochastic differential equation (SDE), $d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w}$, where $\mathbf{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is the drift coefficient, $g(t) \in \mathbb{R}$ is the diffusion coefficient, and w represents the standard Brownian motion. Letting the distribution of $\mathbf{x}(t)$ be $p_t(\mathbf{x})$, the diffusion process is such that $\mathbf{x}(0) \sim p_0$, and $\mathbf{x}(T) \sim p_T$, where p_0 is the data distribution, and p_T is an easy to sample from prior distribution with a tractable form. Samples from the data distribution p_0 are obtained by reversing the diffusion process, which can be described as $d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] dt + g(t)d\overline{\mathbf{w}}$, where $\overline{\mathbf{w}}$ is a Brownian motion in the reverse time direction, and dt represents an infinitesimal negative time step. This reverse SDE can be computed using the drift and diffusion coefficients of the forward SDE, as well as the score of $p_t(x)$ for each $t \in [0, T]$. The score function, $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, may then be used to gather samples from p_0, using those from a prior distribution p_T. Slightly altering this process, we can utilize the Predictor-Corrector method, which is a hybrid approach that combines numerical SDE solving with the Langevin Markov Chain Monte Carlo (MCMC) method. It works by computing $\mathbf{x}_{i+1} = \mathbf{x}_i + \epsilon\nabla_{\mathbf{x}} \log p(\mathbf{x}_i) + \sqrt{2\epsilon}\mathbf{z}_i$ for i from 1 to N, where $\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \epsilon > 0$ is the step size, and $\mathbf{x}_1$ is initialized from any prior distribution $\pi(\mathbf{x}_1)$. As N tends ti infinity, and ϵ tends to 0, $\mathbf{x}_{N+1}$ becomes a sample from $p(\mathbf{x})$. The SSM, $s_\theta(\mathbf{x}, t)$, is trained to approximate $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ using the wighted sum of denoising score-matching objectives,

$$\min_\theta \mathbb{E}_{t\sim\mathcal{V}(0,T)}\left[\lambda(t)\mathbb{E}_{\mathbf{x}(0)\sim p_0(\mathbf{x})}\mathbb{E}_{\mathbf{x}(t)\sim p_{ot}(\mathbf{x}(t)|\mathbf{x}(0))}\left[\left\|s_\theta(\mathbf{x}(t), t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))\right\|_2^2\right]\right],$$

where $\mathcal{V}(0, T)$ is a uniform distribution over [0, T], $p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0))$ denotes the transition probability from $\mathbf{x}(0)$ to $\mathbf{x}(t)$, and $\lambda(t) \in \mathbb{R}_{>0}$ denotes a positive weighting function. A stochastic differential equation for perturbing data distribution, p_0, to a prior distribution, p_T, is defined as $d\mathbf{x} = \sigma^t d\mathbf{w}, t \in [0, 1]$, resulting in $p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}\left(\mathbf{x}(t); \mathbf{x}(0), \frac{1}{2\log\sigma}(\sigma^{2t} - 1)\mathbf{I}\right)$. The weighting function, $\lambda(t)$, is defined as $\frac{1}{2\log\sigma}(\sigma^{2t} - 1)$. When σ is large, the prior distribution, $p_{t=1}$, becomes

$\int p_0(\mathbf{y})\mathcal{N}\left(\mathbf{x};\mathbf{y},\frac{1}{2\log\sigma}\left(\sigma^2-1\right)\mathbf{I}\right)d\mathbf{y} \approx \mathbf{N}\left(\mathbf{x};\mathbf{0},\frac{1}{2\log\sigma}\left(\sigma^2-1\right)\mathbf{I}\right)$, which is approximately independent of the data distribution and is easy to sample from. Sampling can be performed using SDEs or with Ordinary Differential Equations (ODE). For any SDE of the form $d\mathbf{x} = \mathbf{f}(\mathbf{x},t)dt + g(t)d\mathbf{w}$, there exists a corresponding ODE, $d\mathbf{x} = \left[\mathbf{f}(\mathbf{x},t) - \frac{1}{2}g(t)^2\nabla_{\mathbf{x}}\log p_t(\mathbf{x})\right]dt$, such that their trajectories have the same marginal probability density $p_t(\mathbf{x})$. By first sampling from from p_T, an integrating $d\mathbf{x} = -\frac{1}{2}\sigma^{2t}s_\theta(\mathbf{x},t)dt$ from T to 0 in the reverse-time direction, we can obtain a sample from p_0.

3 Methodology

Two different architectures were implemented based on existing papers [1, 2], using Jupyter Notebook [13] and PyTorch [14]. Both models were initially trained on Paperspace [15], which offers similar tiers to Google Colab [16], while providing a better choice of devices. NVIDIA RTX A6000 24GB and NVIDIA A100 80GB were used interchangeably, depending on availability. The CIFAR-10 [3] training set contained 49,984 images, which were partitioned into 781 batches of 64 images each. Images were of size 3x32x32. Considering the DDPM model with CIFAR-10, training took roughly 1:20 minutes, whereas sampling around 30 seconds, bringing a single epoch run time to approximately two minutes. Considering the SMM with the same data, training took roughly 20 seconds, whereas sampling around 35 seconds, bringing a single epoch run time to approximately 1 minute. The FFHQ [4] training set contained 70,000 images, which were partitioned into 1094 batches of 64 images each, apart from the last one, containing 48 images. Images were resized from 3x128x128 to 3x96x96, Data was downloaded from the official repository and stored in 70 folders with 1,000 images each (apart from the last one, again). Considering the DDPM model with FFHQ, training took roughly 15 minutes, whereas sampling around 2 minutes, bringing a single epoch run time to approximately 17 minutes. Considering the SMM with the same data, training took roughly 6:15 minutes, whereas sampling around 1:50 minutes, bringing a single epoch run time to approximately 8 minutes. Times were empirically observed on A100 80GB, and were slightly longer on RTX 6000. The T variable was set to $1,000$ for the DDPM model, and to $4,000$ for the SMM. Loss was calculated per each batch and the results were gathered to compute the mean average loss over every epoch. Losses were observed to slowly but steadily decrease over time.

3.1 Denoising Diffusion Probabilistic Model (DDPM)

During the training process, batches of 64 images were loaded and a random value of t was generated for each sample in the batch, representing the time step at which the noise reduction will be performed during the diffusion process. Noise, ϵ, was sampled from the Normal Distribution. As a result of the forward diffusion process, a denoised version of the x_0 input image, x_t, was generated with the $q(x_t|x_0)$ function. The output of the the $q(x_t|x_0)$ function is obtained by computing $mean + \sqrt{variance} * noise$

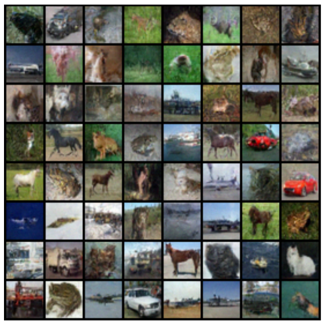
A neural network, $\epsilon_\theta(x_t,t)$, was used to compute $\epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)$, which is the estimated noise. The loss was then computed between actual noise,ϵ, and the estimated noise, ϵ_θ. The network structure was based on the U-Net [17] architecture. Using sinusoidal position embedding technique, the input was first projected into a feature map of shape [64, 64, 32, 32] in case of CIFAR-10, and [64, 64, 96, 96] in case of FFHQ, by passing the input through two convolutions and using the Swish activation. The embedding was then passed through a down-sampling layer (encoder), a bottleneck layer, and the up-sampling layer (decoder). The final convolution was then performed on the normalised output, activated with the Swish function. A Residual block applied two 2D convolutions, using Swish activation, group normalisation with 32 groups each, and skip connections. The down-sampling layer consists of four such blocks, combined with attention using 2 heads. The Bottleneck consists of two Residual blocks with attention in the middle, and is applied right after the down-sampling process, to the lowest resolution of the U-Net. The up-sampling layer has a similar structure to the down-sampling layer and reverses the encoding process.

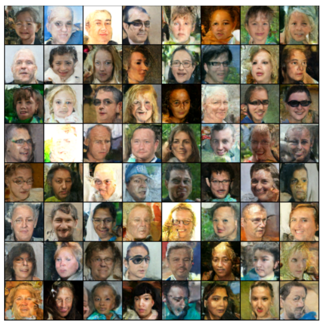
The sampling process generated a batch of 64 new images by removing noise from a given input image in a series of T steps. The input image of the same shape as training images, x_T, was first generated from the Normal Distribution. At each step, a new image is generated from the previous one. Noise was first calculated with the trained $\epsilon_\theta(x_t, t)$ network. Next, α_t and $\bar{\alpha}_t$ values were calculated, which were then used to compute the mean and variance of the Gaussian distribution used for sampling. The result was a new image with less noise than the previous one, with the final image containing no noise.

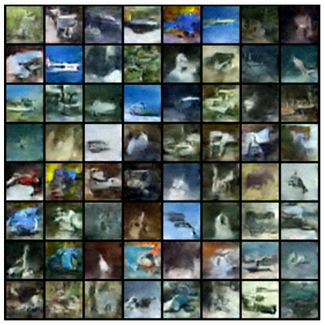
3.2 Score-matching Model (SMM)

Batches of 64 images were loaded and a random value of t was generated for each image in the batch. Noised image was produced by adding the noise scaled by the standard deviation to the original images. The score was then calculated using a deep neural network. Loss was computed using the mean of the sum of squares of the score multiplied by the standard deviation, added to the noise and squared again, over the spatial dimensions of the image. This process was repeated for each epoch and checkpoints were saved every few. U-Net architecture was also used to construct the SMM, consisting of an encoder and a decoder. The encoder takes the input image and passes it through a series of convolutions, each of which followed by a dense layer and a group normalization. The decoder takes the embedding and upscales it through a series of convolutions. The output of each layer is passed through a dense layer and group normalization before being added to the output of the previous layer. The final output is a denoised version of the input image. The network was implemented in such way, that is allows a choice between the ODE- and SDE-based sampling, with the latter offering both, Euler-Maruyama only sampler, and Euler-Maruyama with Langevin MCMC, referred to as Sample-Corrector (SC). The ODE-based sampling with SC approach was used for all training.

4 Results







The DDPM model was slower to train and to sample from, but produced samples of perceivable higher quality with less epochs required. Parameter T was found to be the most detrimental for sampling speed, with higher values meaning more denoising steps and thus, longer sampling time. Generally, it is still not well-understood how much noise is enough. Franzese et al. [18] provides an interesting analysis of this topic. In images produced by the DDPM trained on CIFAR-10, shapes and objects were clearly recognisable. The images produced by SMM trained on CIFAR-10 are not as clear, but this is likely due to the relatively short training time. It is believed that with more epochs, resulting samples would heve been of much higher quality. Same was the case with FFHQ data set. Training had to be stopped due to limited time and compute resources, but diffusion-based models have excellent convergence properties, and it is expected that longer training would results in more realistic, higher quality samples. In all cases however, images were truly diverse. Above are samples generated by DDPM after 940 epochs of training on CIFAR-10, samples generated by DDPM after only 18 epochs of training on FFHQ, and samples generated by SMM after 2020 epochs of training on CIFAR-10 respectively.

5 Discussion (Limitations and Conclusions)

Hyperaprameters, such as the learning rate were chose arbitrarily. Tuning them with Optuna [19] could considerably improve the results. Some sort of guiding could also be implemented for training on CIFAR-10. It could be beneficial, if we ensured that the batches contained images representing every class available, in a roughly equal ratio. Nichol and Dhariwal [20], used exponential smoothing and Cosine β scheduling in their follow up to [1]. These ideas could also be explored further. Models could obviously be trained for longer.

Bonuses

This submission has a total bonus of +6 marks, as it was trained not only on CIFAR-10, but also on FFHQ data set images, resized to 96x96.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

[2] Yang Song et al. "Score-based generative modeling through stochastic differential equations". In: *arXiv preprint arXiv:2011.13456* (2020).

[3] Alex Krizhevsky and Geoffrey Hinton. *CIFAR-10 (Canadian Institute For Advanced Research)*. 2009. URL: `https://www.cs.toronto.edu/~kriz/cifar.html`.

[4] NVIDIA (NVlabs). *Flickr-Faces-HQ Dataset*. Accessed: 2023-01-10. 2019. URL: `https://github.com/NVlabs/ffhq-dataset`.

[5] Phil Wang. *Denoising Diffusion PyTorch*. Version v0.1.1. Accessed: 2023-01-12. 2021. URL: `https://github.com/lucidrains/denoising-diffusion-pytorch`.

[6] Niels Rogge and Kashif Rasul. *Annotated Diffusion: Efficiently Training Transformers on Long Sequences*. 2021. arXiv: `2110.13952 [cs.LG]`.

[7] Google. *Example of Using a Diffusion Model for Image Generation in Colab*. Accessed on: ¡insert date¿. 2022. URL: `https://colab.research.google.com/drive/120kYYBOVa1i0TD85RjlEkFjaWDxSFUx3?usp=sharing#scrollTo=XCR6m0HjWGVV`.

[8] LabML. *LabML: Experiments and research papers on artificial intelligence*. Accessed on 2023-02-06. URL: `https://nn.labml.ai/`.

[9] Yang Song. *Generative Modeling by Estimating Gradients of the Data Distribution*. Accessed: 2022-12-17. 2021. URL: `%5Curl%7Bhttps://yang-song.github.io/blog/2021/score/%7D`.

[10] Michael Dome. *Diffusion-Models-pytorch*. Accessed on 2023-01-17. 2021. URL: `https://github.com/dome272/Diffusion-Models-pytorch`.

[11] Niels Rogge and Kashif Rasul. *Annotated Diffusion*. Accessed on 2023-01-19. 2021. URL: `https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/annotated_diffusion.ipynb`.

[12] Aapo Hyvärinen and Peter Dayan. "Estimation of non-normalized statistical models by score matching." In: *Journal of Machine Learning Research* 6.4 (2005).

[13] The Jupyter Development Team. *Project Jupyter*. 2015–. URL: `https://jupyter.org`.

[14] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. URL: `https://pytorch.org`.

[15] *Paperspace - Cloud Machine Learning, AI, and effortless GPU infrastructure*. URL: `https://www.paperspace.com`.

[16] Google. *Google Colaboratory*. URL: `https://colab.research.google.com`.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241.

[18] Giulio Franzese et al. “How much is enough? a study on diffusion times in score-based generative models”. In: *arXiv preprint arXiv:2206.05173* (2022).

[19] Takuya Akiba et al. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. URL: `https://optuna.org/`.

[20] Alexander Quinn Nichol and Prafulla Dhariwal. “Improved denoising diffusion probabilistic models”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8162–8171.